University of Pittsburgh

Algorithms and Data Structures 2
CS 1501

Spring 2023

Sherif Khattab

ksm73@pitt.edu

(Slides are adapted from Dr. Ramirez's and Dr. Farnan's CS1501 slides.)

Announcements

- Upcoming Deadlines
  - Homework 5: this Friday @ 11:59 pm
  - Lab 4: Tuesday 2/21 @ 11:59 pm
  - Assignment 1: Friday 2/17 @ 11:59 pm

Previous lecture

- Digital Search Tree (DST)
- Radix Search Trie (RST)

This Lecture

- multi-way Radix Search Trie
- De La Briandais Trie

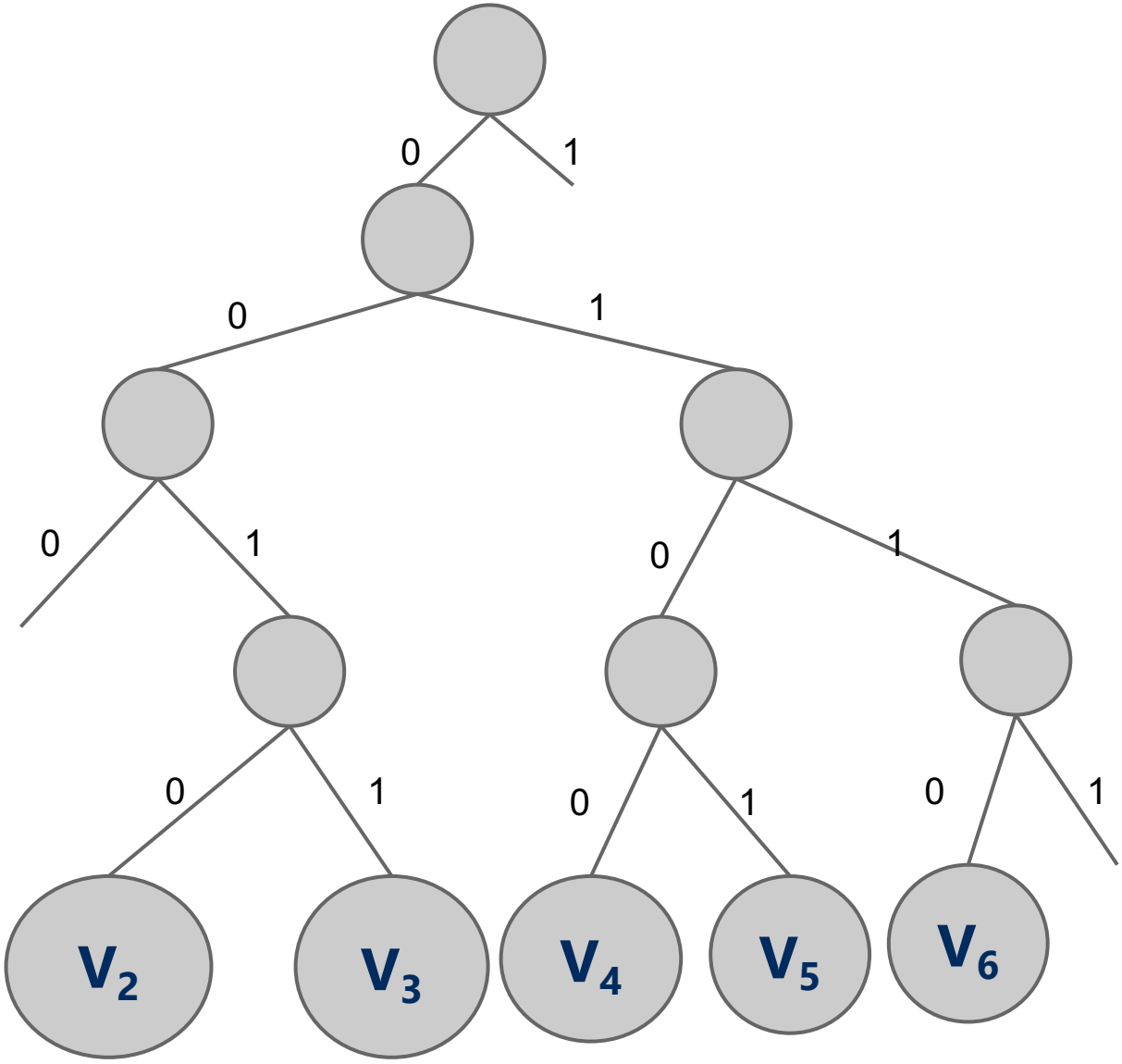
Searching in Radix Search Trie (RST)

- Input: key
- current node ← root
- for each *bit* in the key
  - if current node is null, return *key not found*
  - if bit == 0,
    - current ← current.left
  - if bit == 1,
    - current ← current.right
- if current node is null or the value inside is null
  - return key *not found*
- else return current.value

RST example

Search:

3 0011

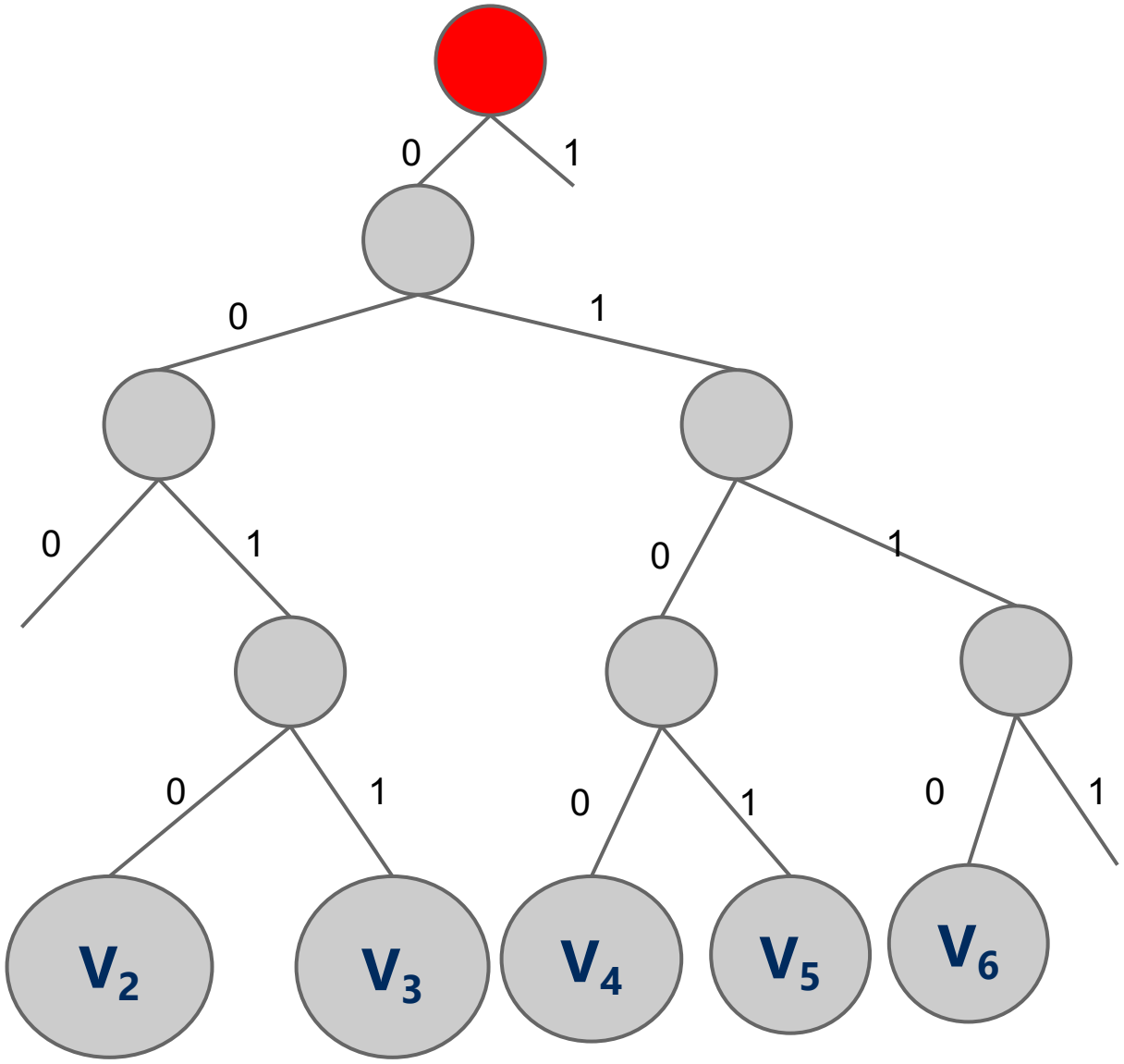


RST example

Search:

3 0011

7 0111

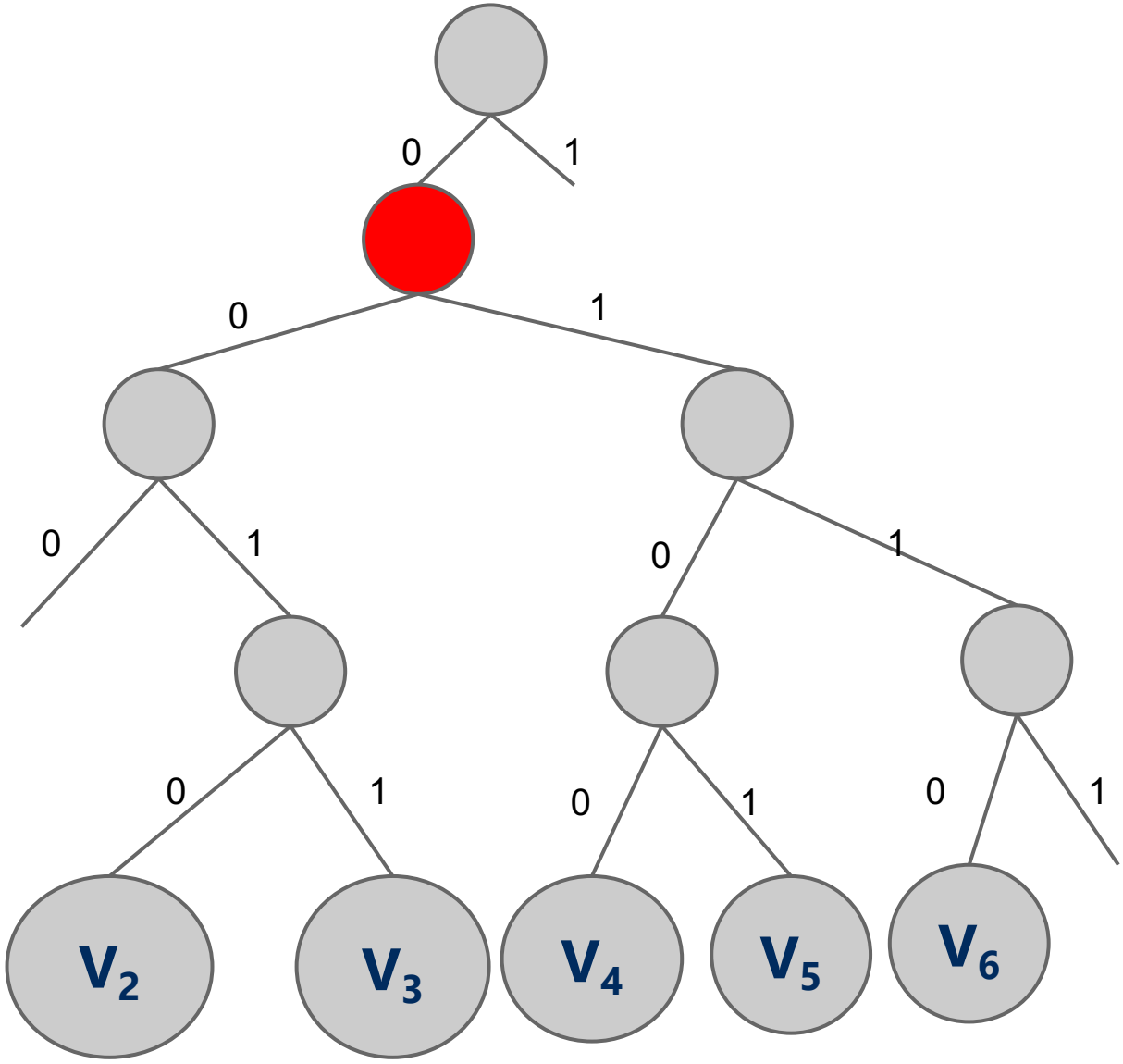


RST example

Search:

3 0011

7 0111

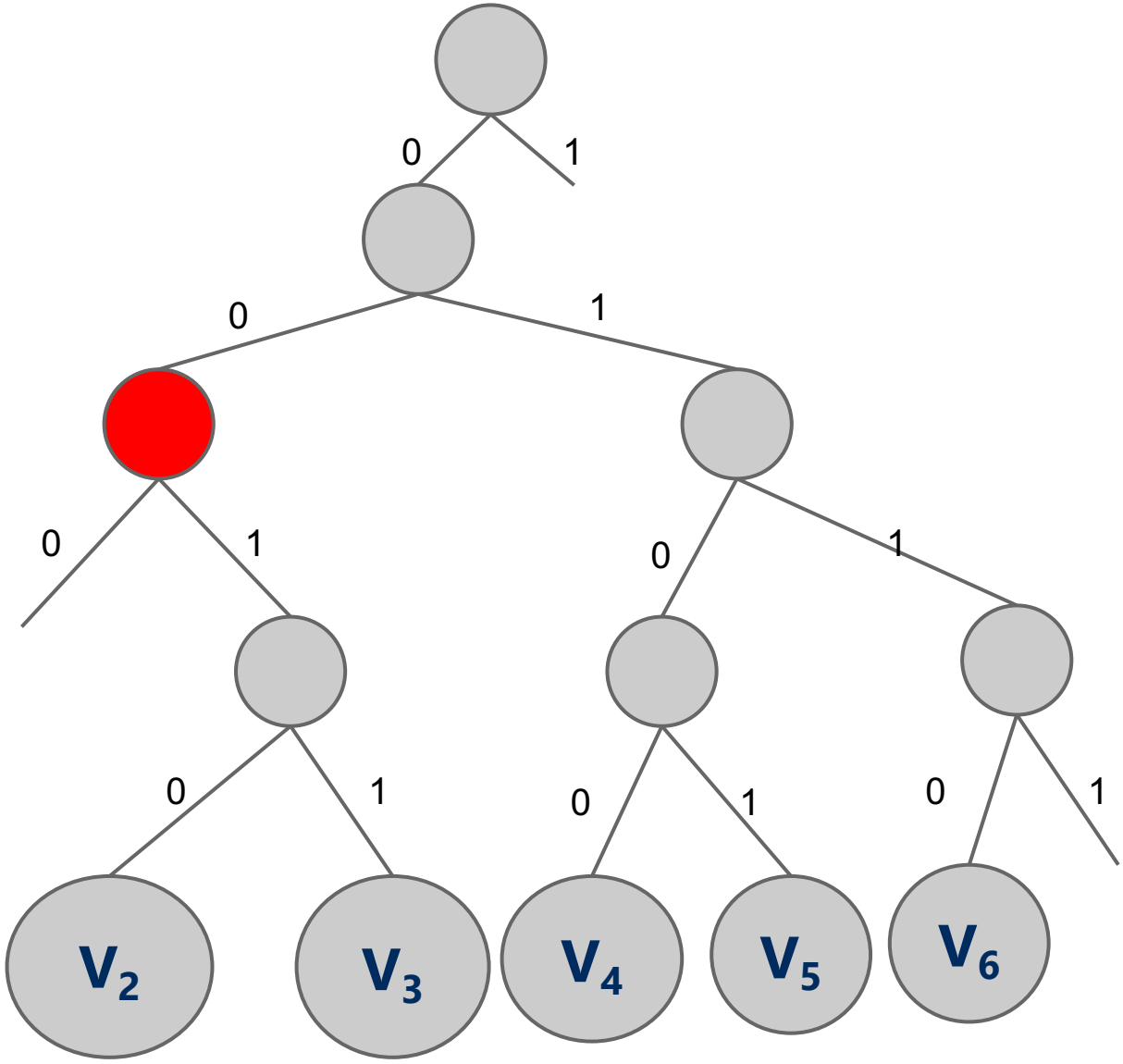


RST example

Search:

3 0011

7 0111

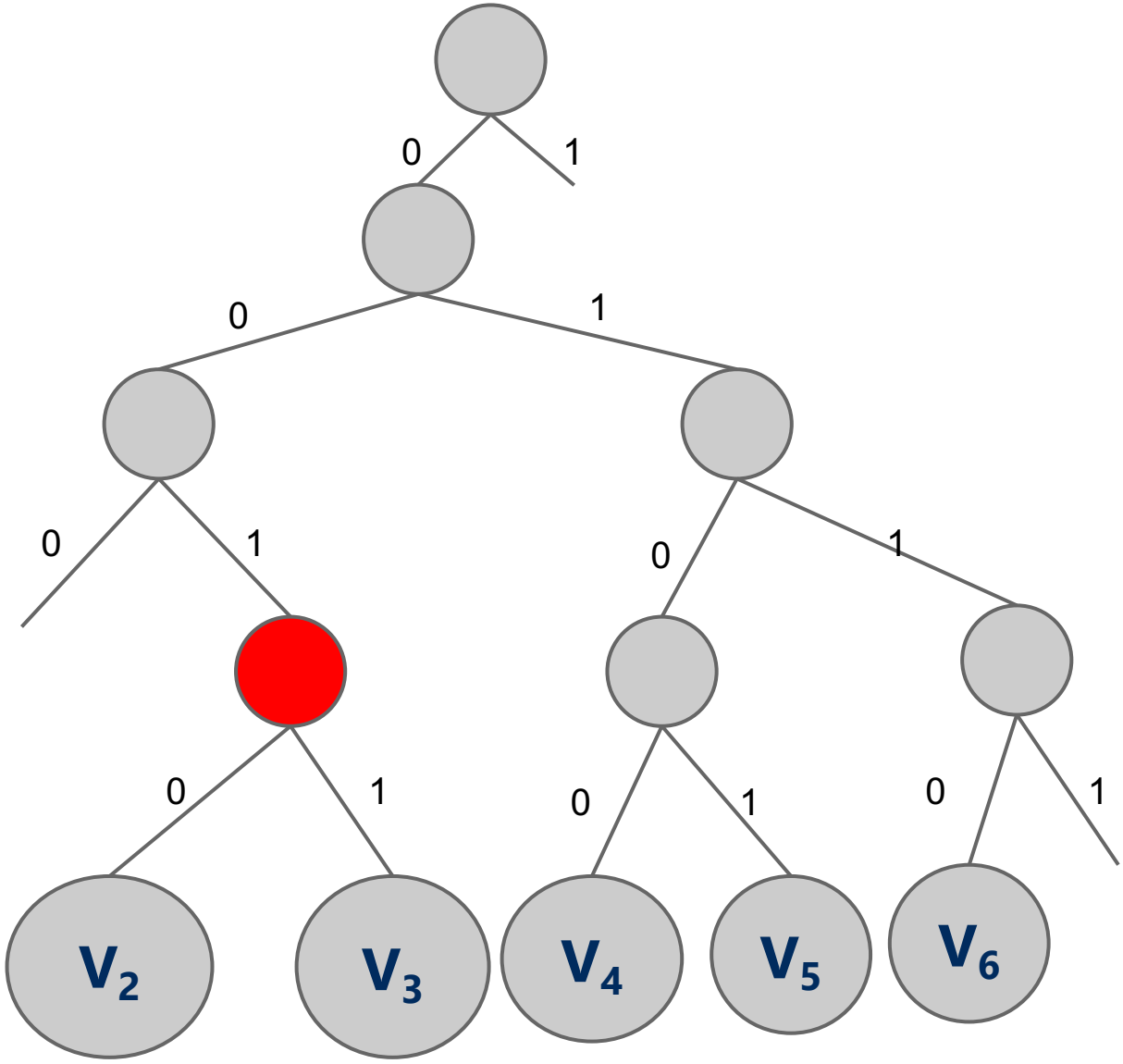


RST example

Search:

3 0011

7 0111

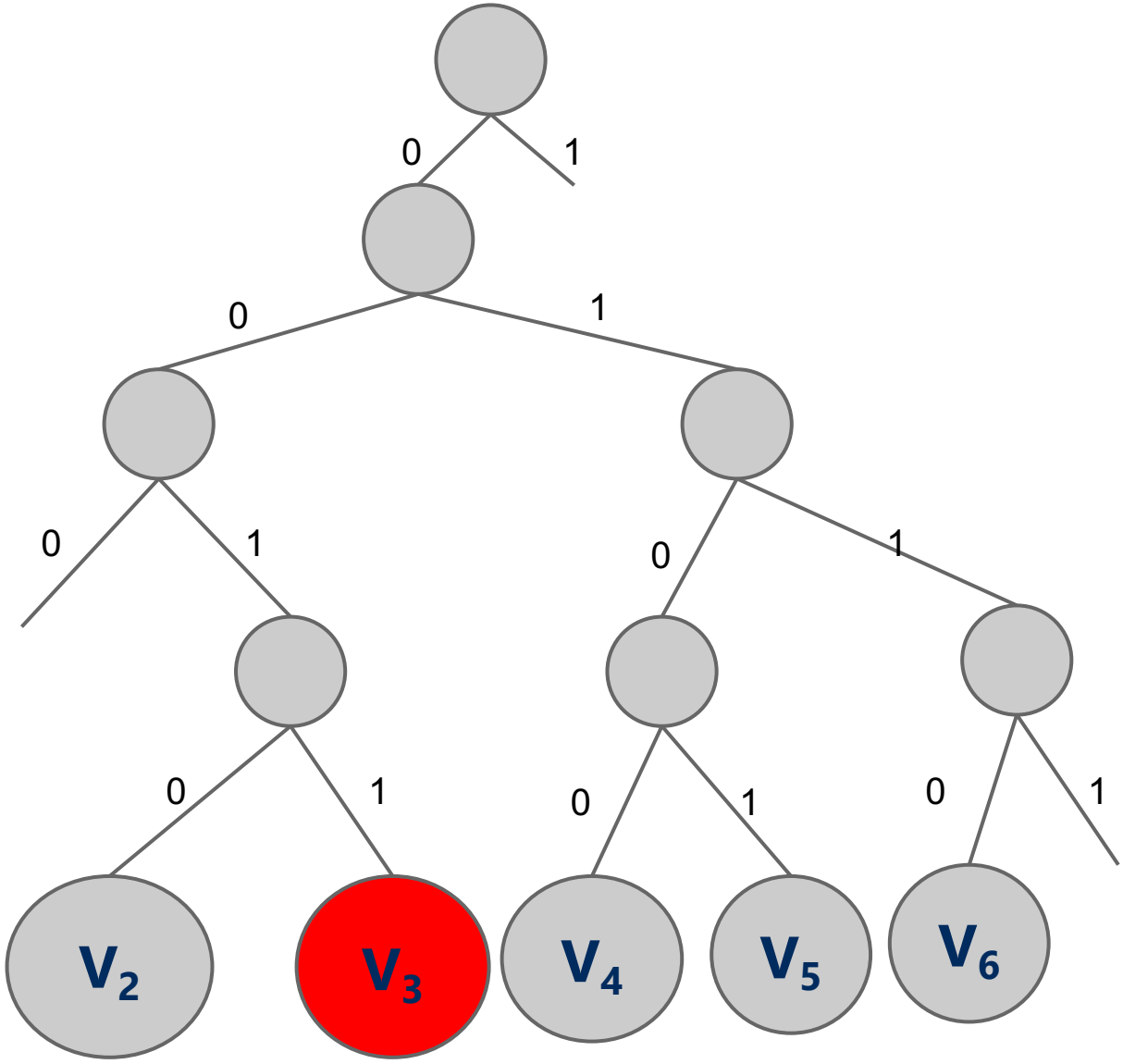


RST example

Search:

3 0011

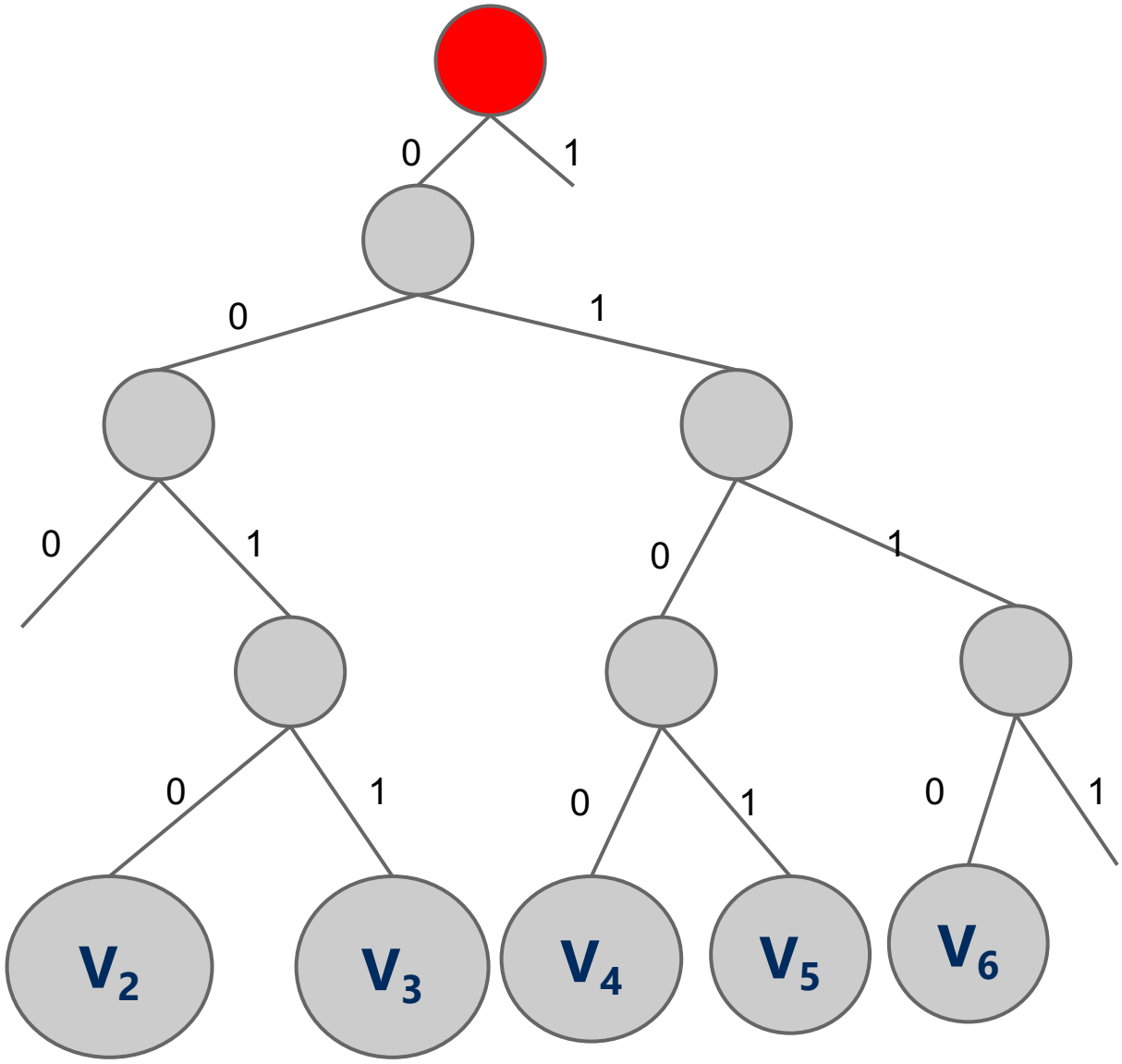
7 0111



RST example

Search:

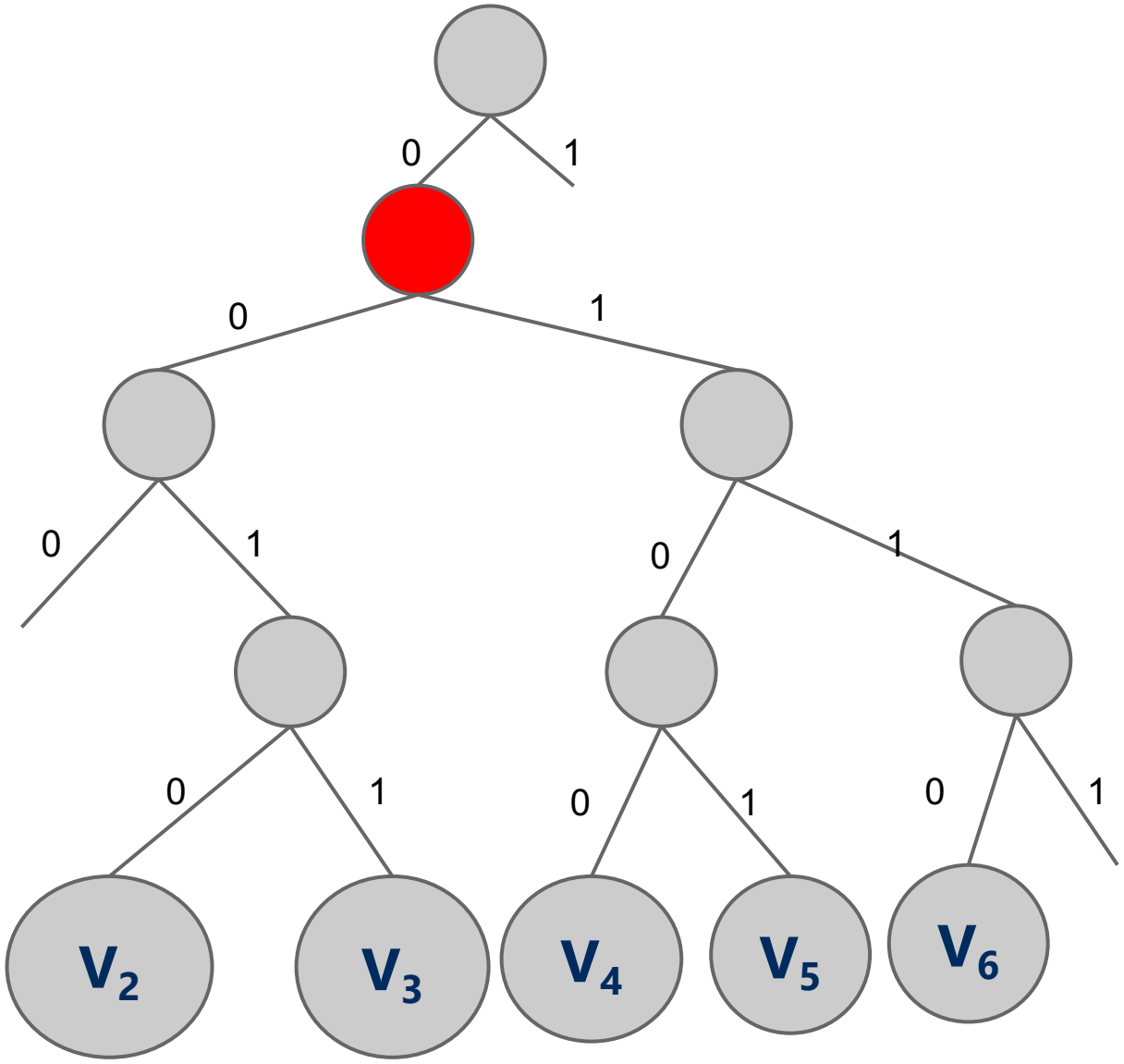
7 0111



RST example

Search:

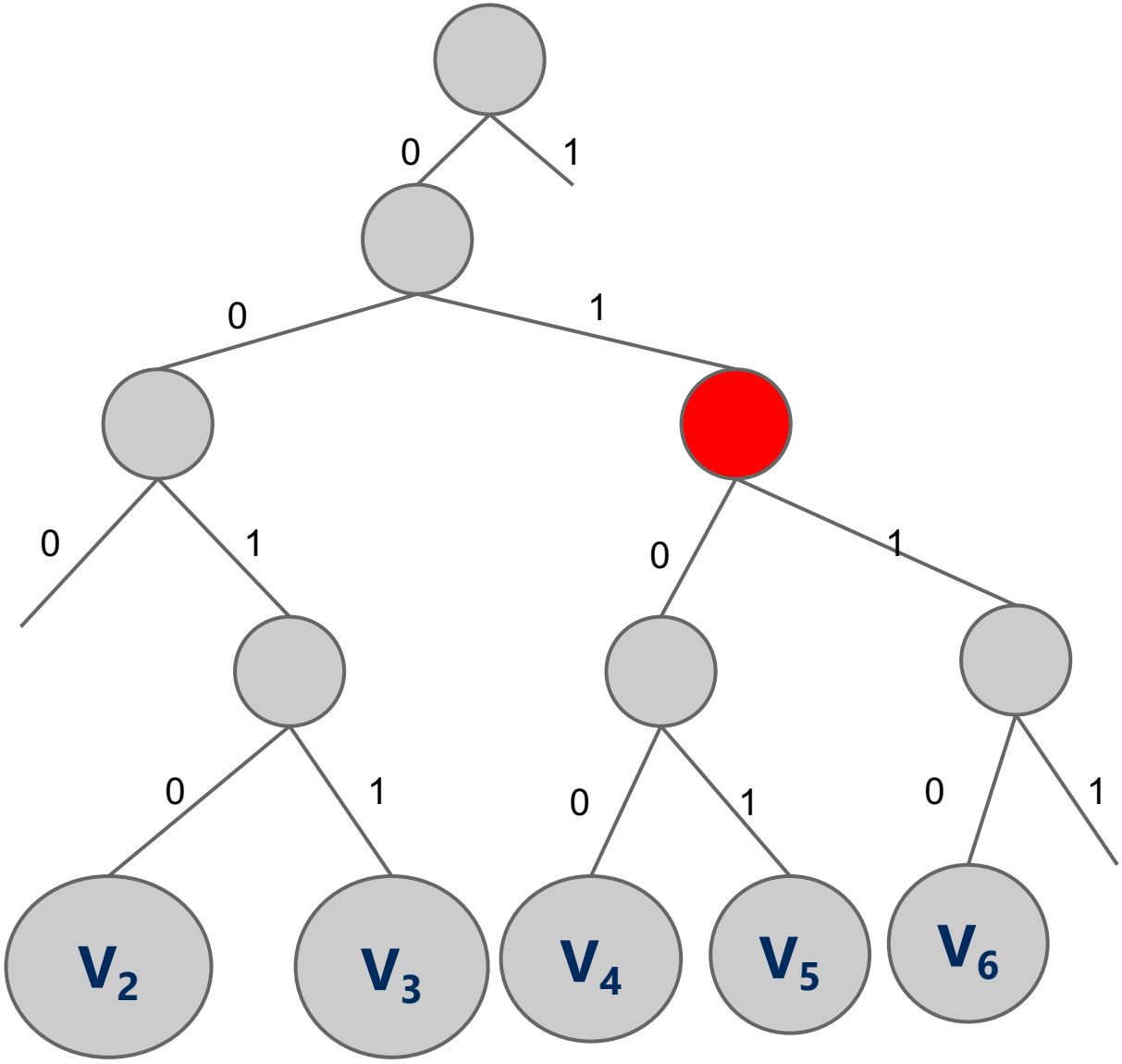
7 0111



RST example

Search:

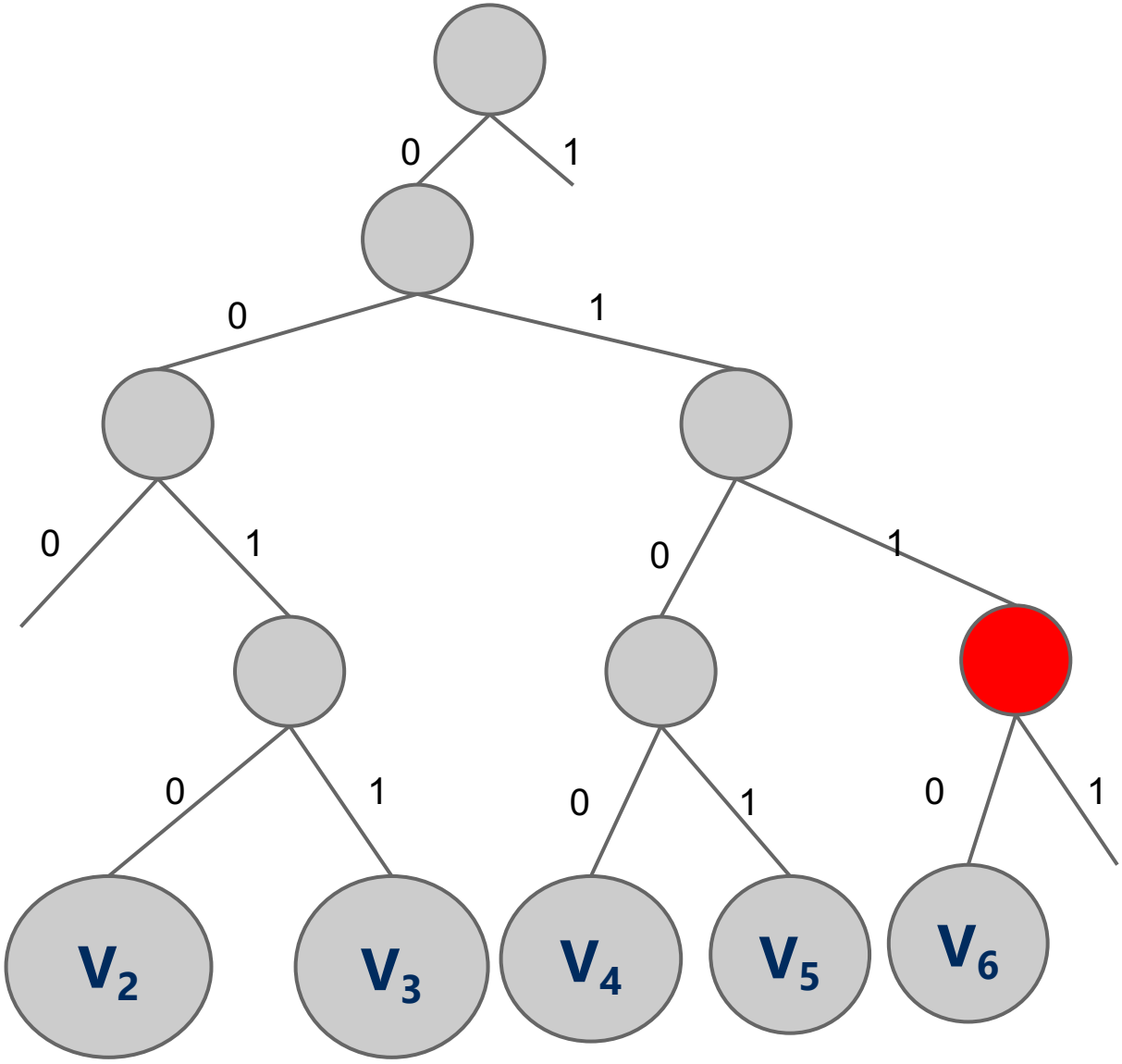
7 0111



RST example

Search:

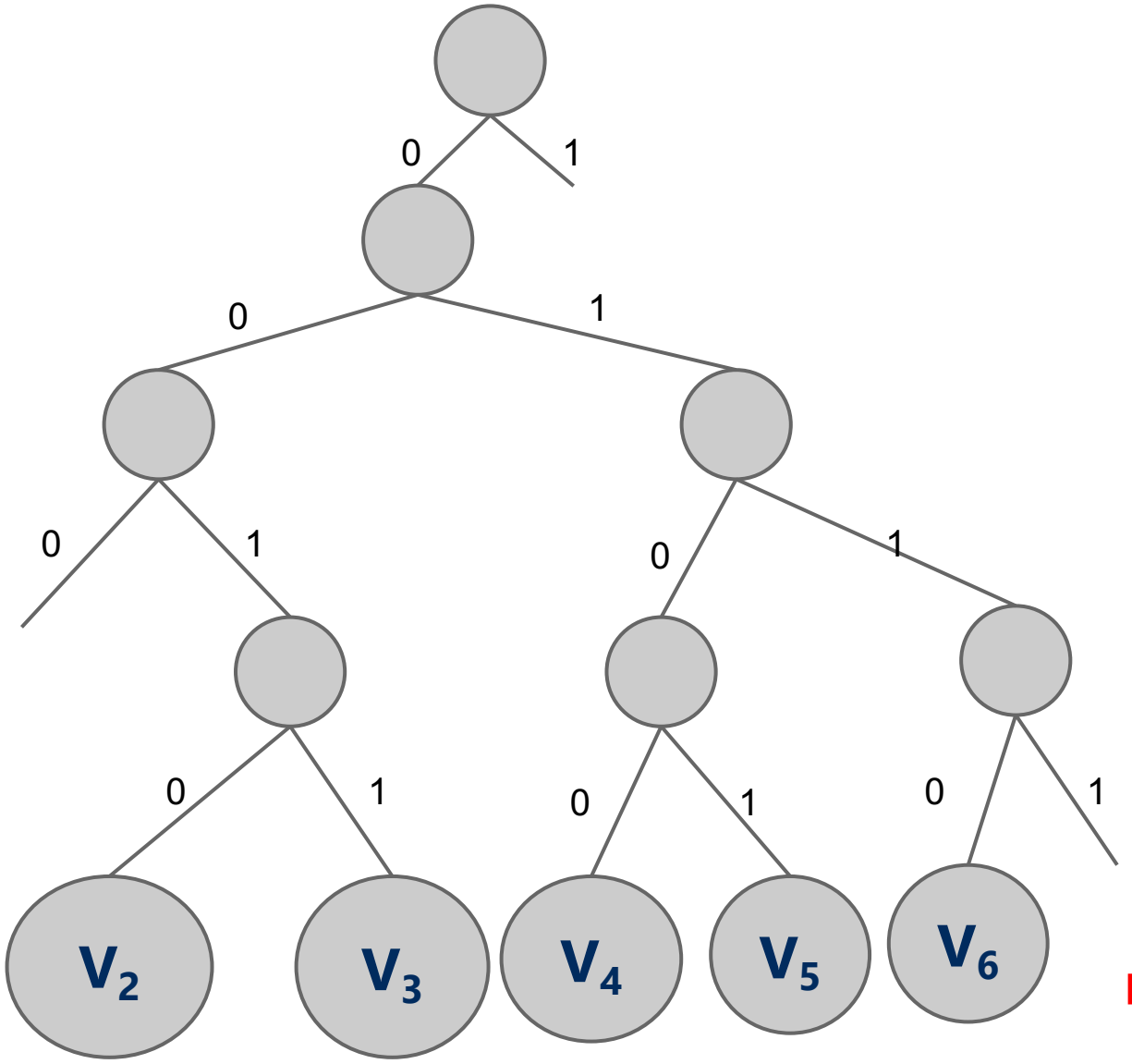
7 0111



RST example

Search:

7 0111



RST Runtime analysis

- **for add:**
  - Theta(b): b is the bit length of the key
  - However, this time we don't have full key comparisons
- for search:
  - **search hit**
    - Theta(b)
  - **search miss**
    - maybe less than Theta(b)

RST Limitations

- Would this structure work as well for other key data types?
- Characters?
  - Characters are the same as 8-bit ints (assuming simple ascii)
- Strings?
- May have huge bit lengths
- How to store Strings?

Larger branching factor tries

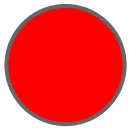
- In our binary-based Radix search trie, we considered one bit at a time
- What if we applied the same method to characters instead of bits in a string?
  - What would this new structure look like?
  - How many children per node?
    - up to R (the alphabet size)
  - Also called R-way radix search tries

Adding to R-way Radix RST

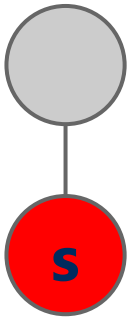
- if root is null, set root ← new node
- current node ← root
- for each *character c* in the key
  - Find the c*th* child of current
    - if child is null, create a new node and attach as the c*th* child
    - move to child
      - current ← child
- insert value into current node

Another trie example

she

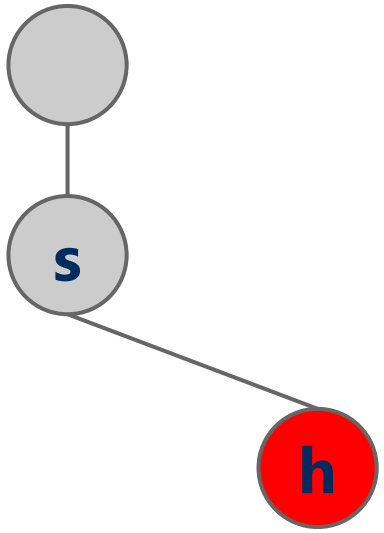


Another trie example



she

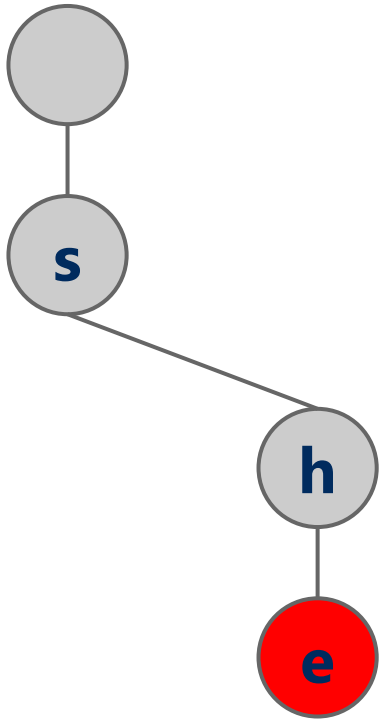
Another trie example



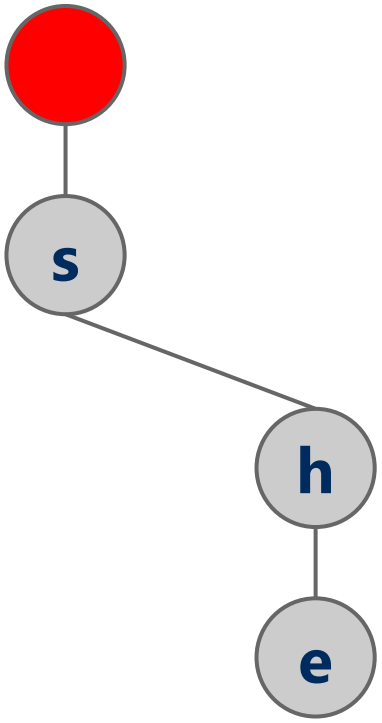
she

Another trie example

she

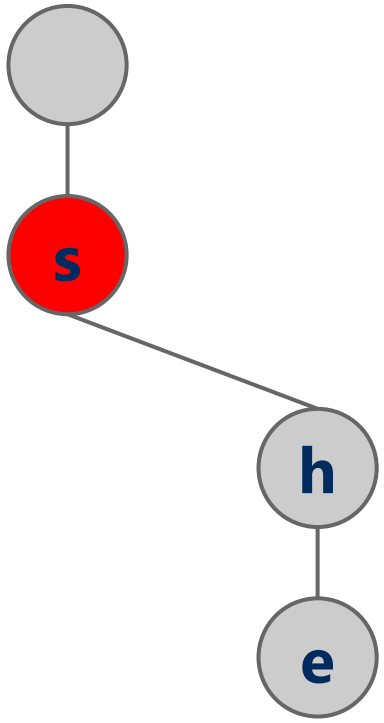


Another trie example



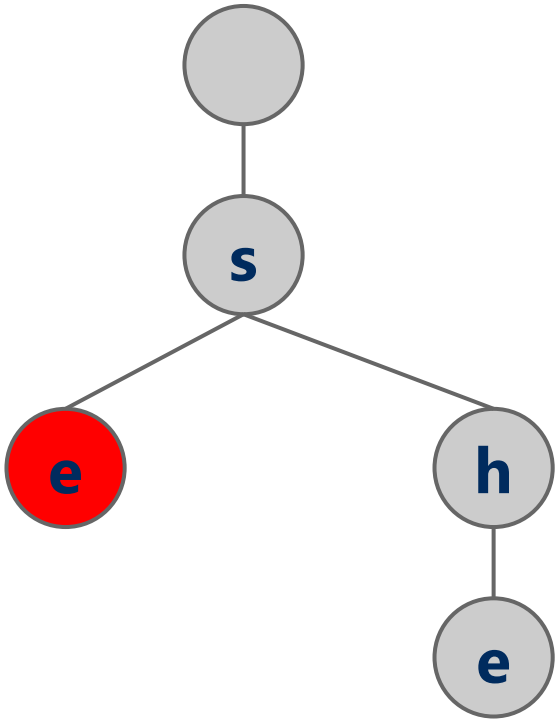
sells

Another trie example



sells

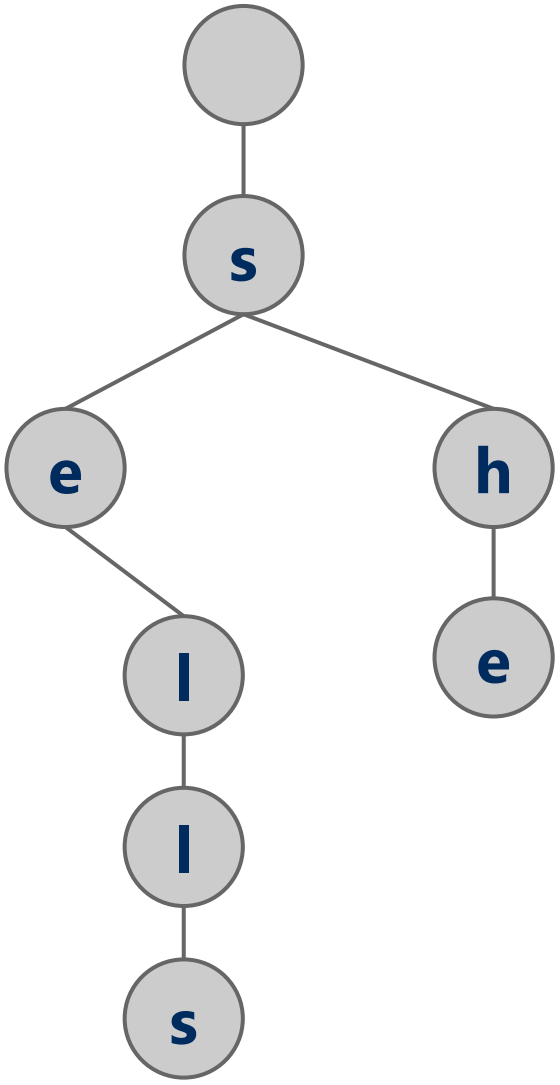
Another trie example



sells

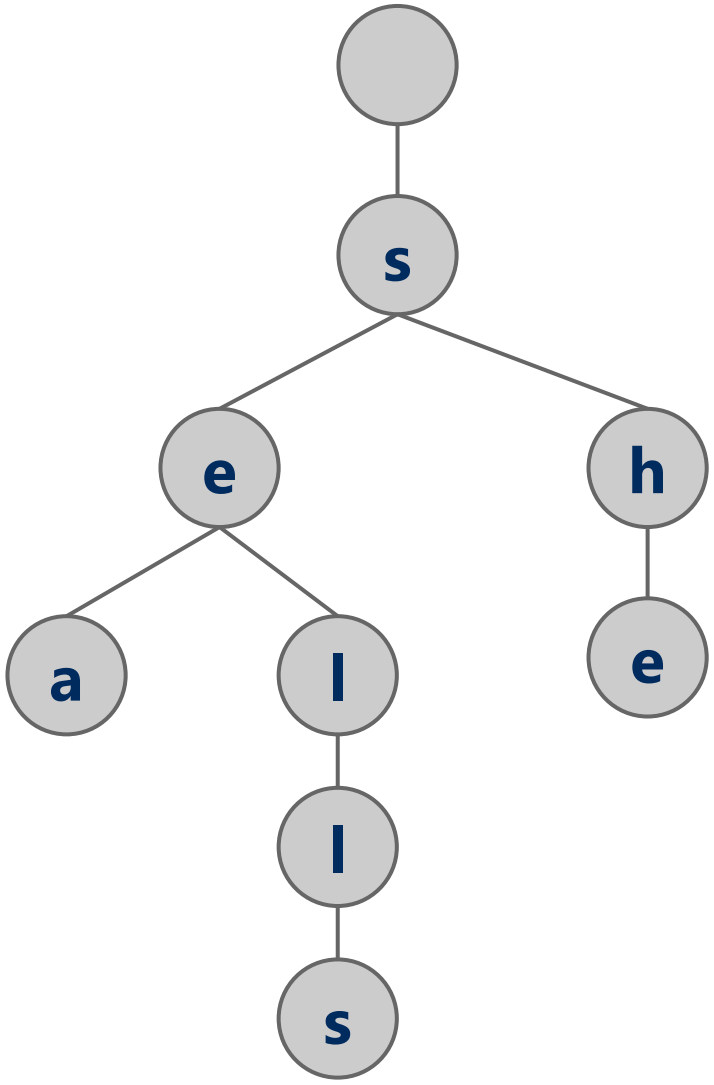
Another trie example

sells



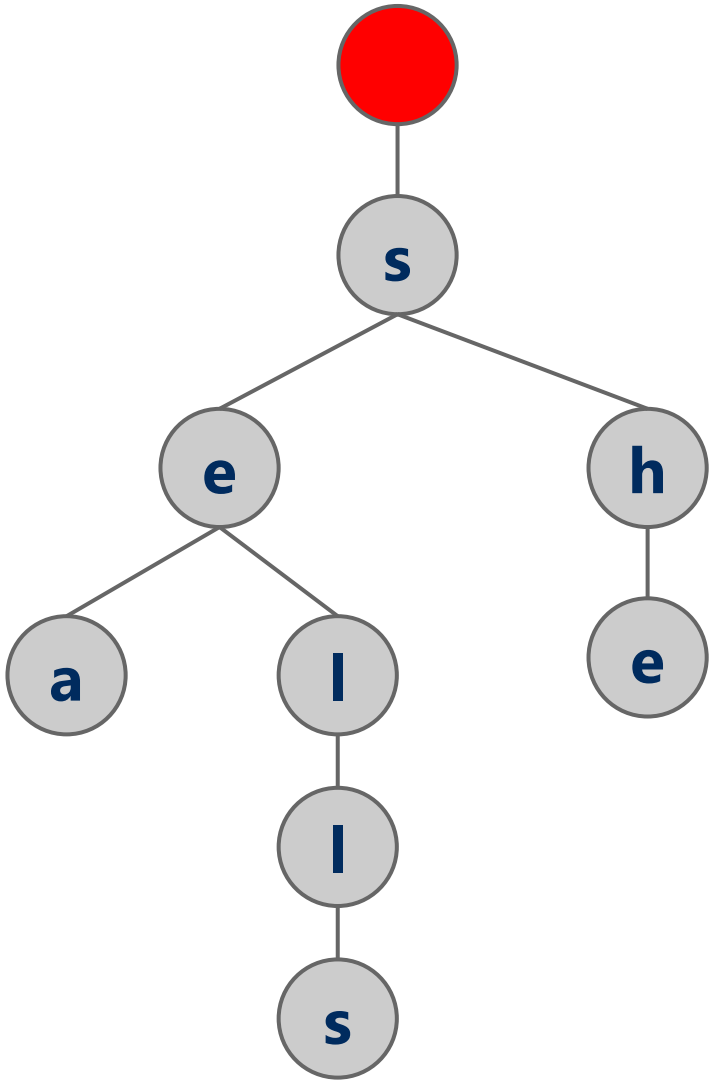
Another trie example

sea



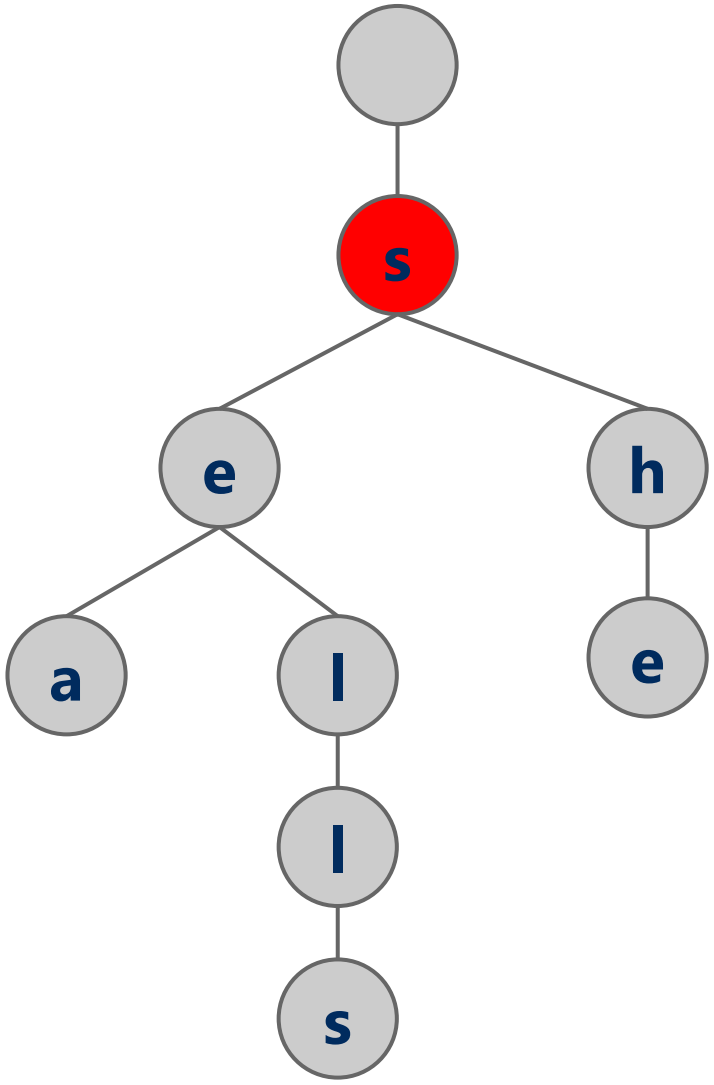
Another trie example

shells



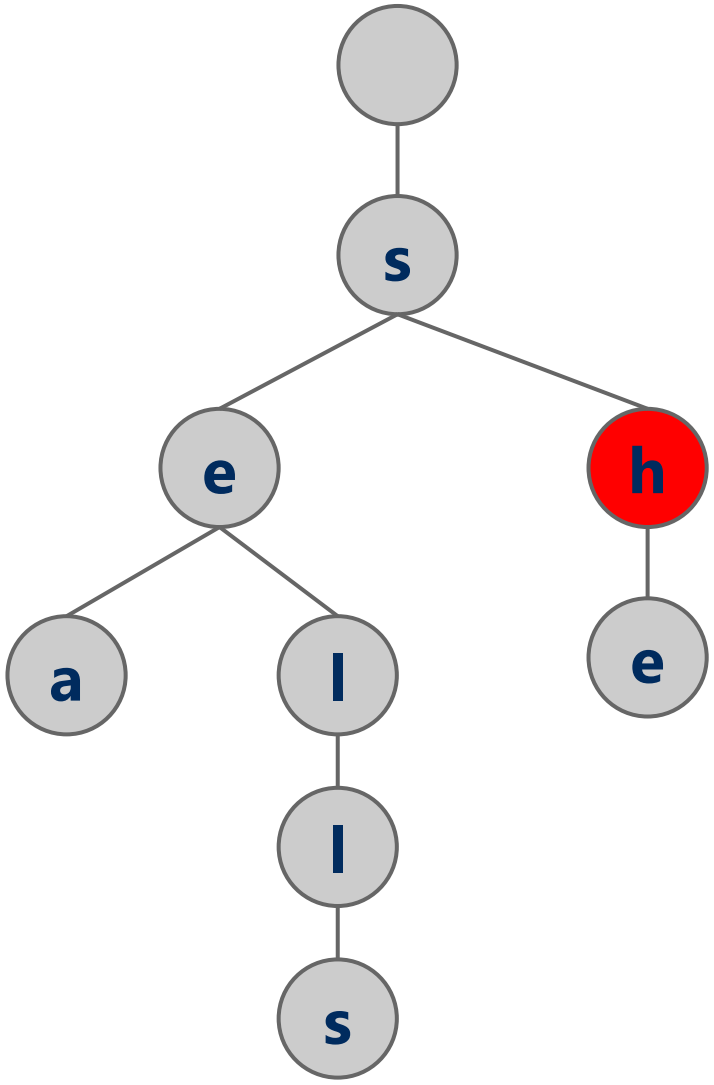
Another trie example

shells



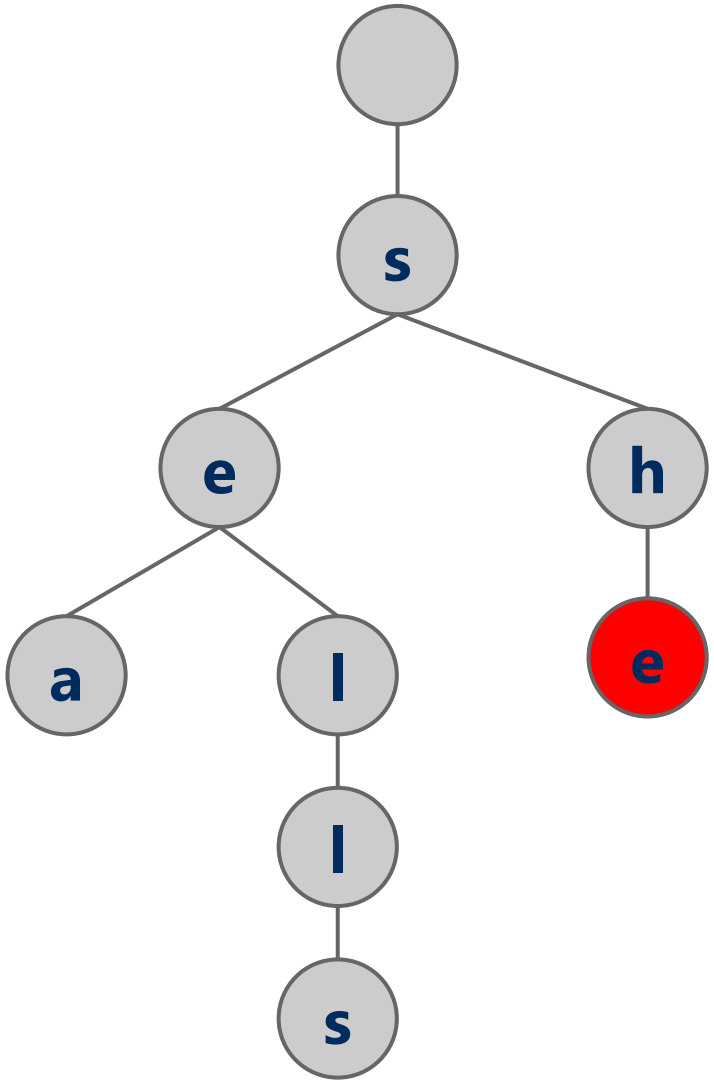
Another trie example

shells



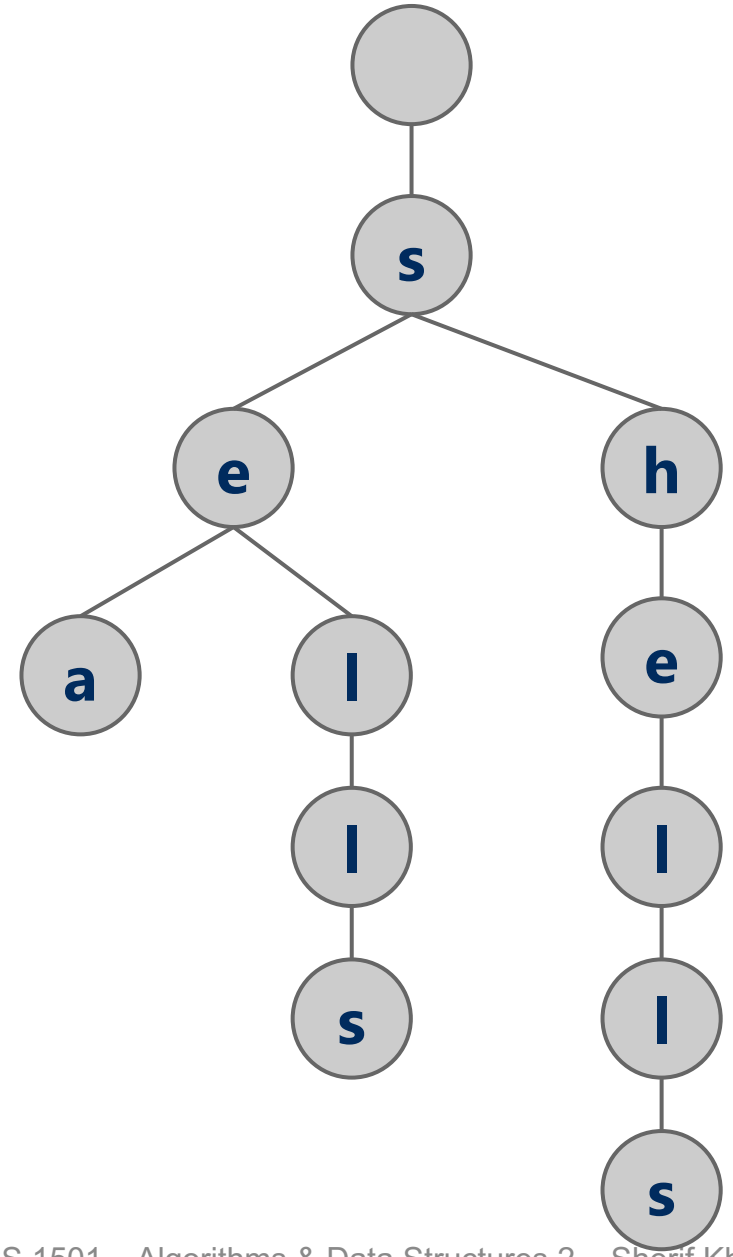
Another trie example

shells

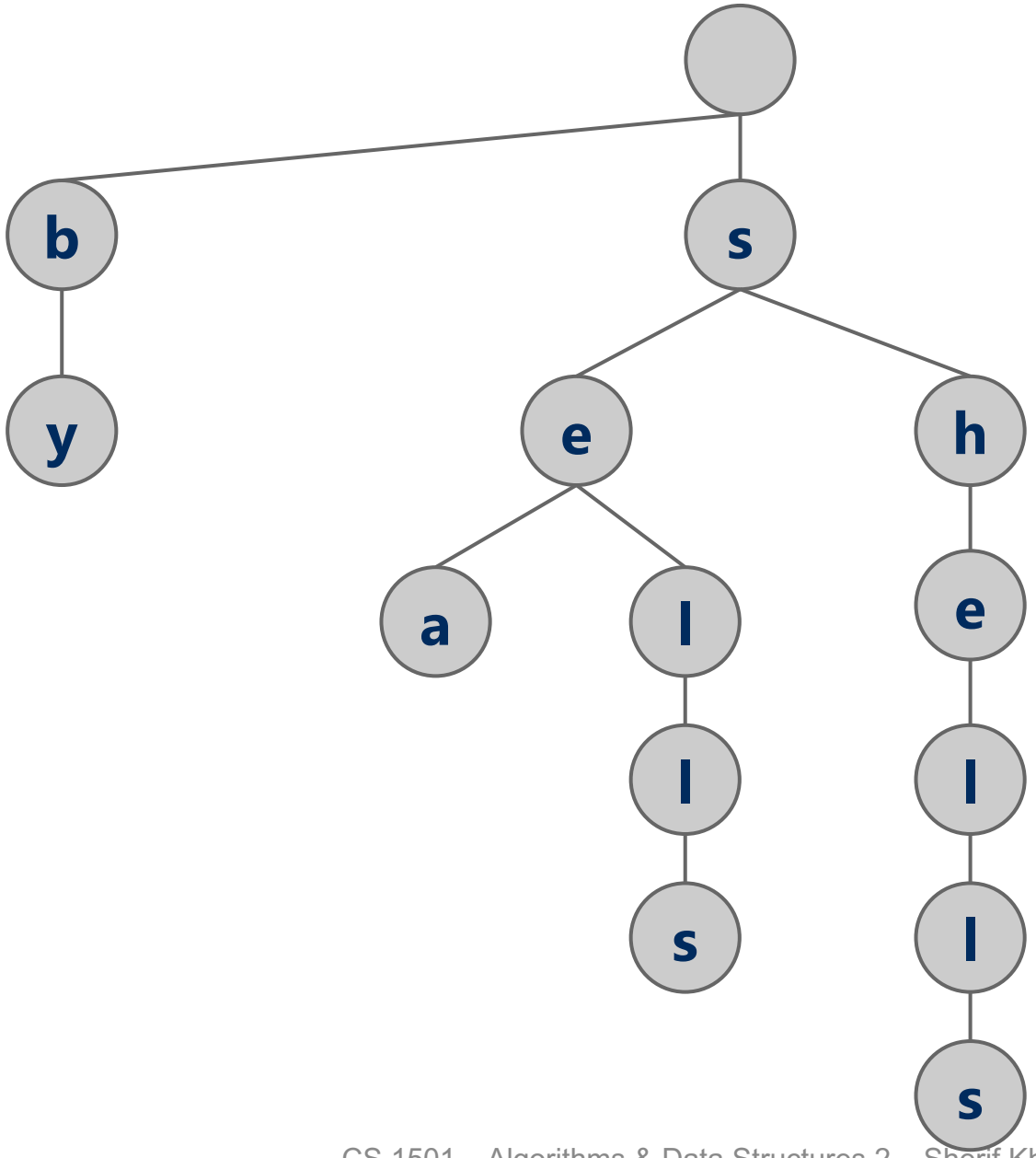


Another trie example

shells

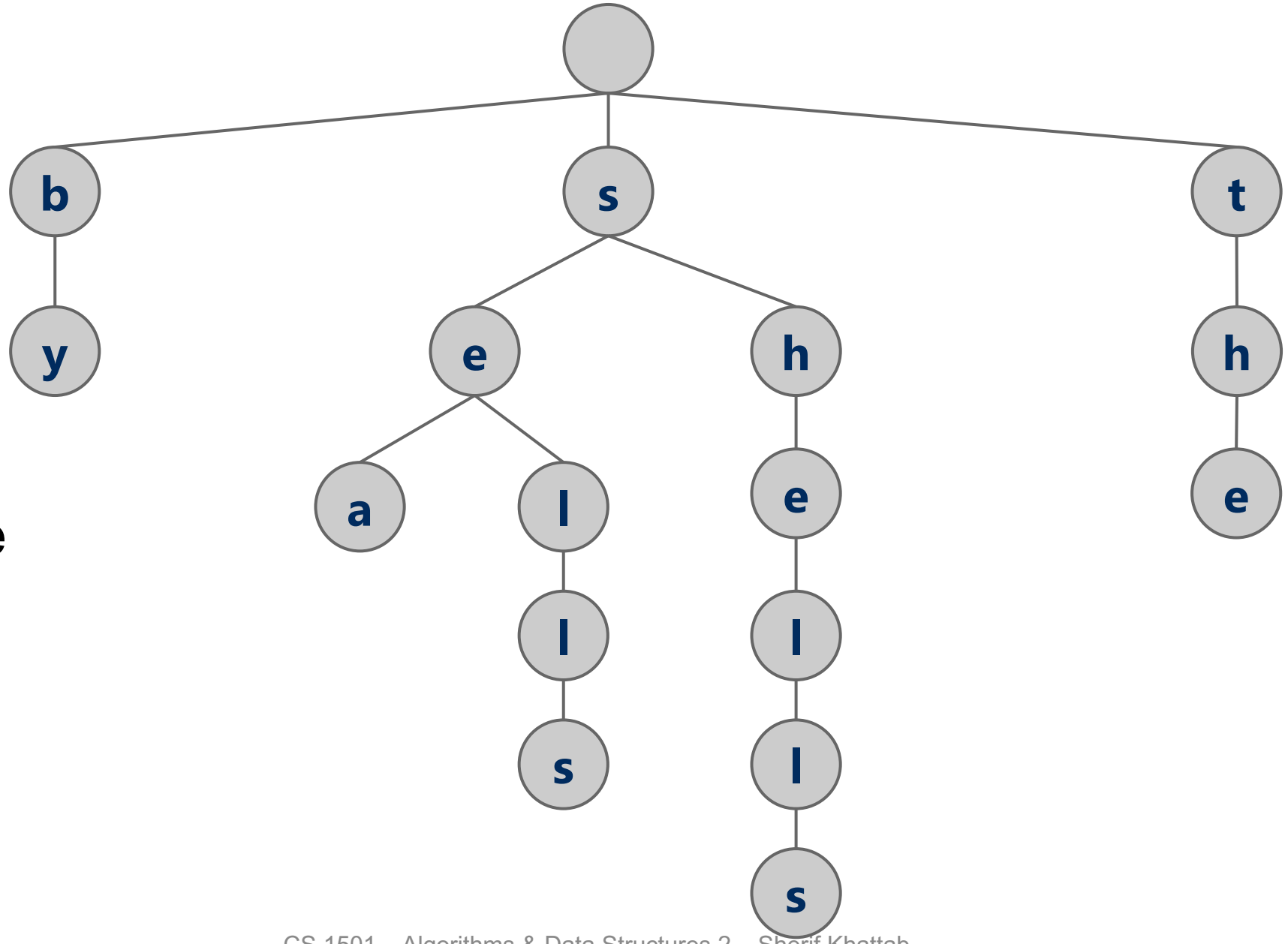


Another trie example



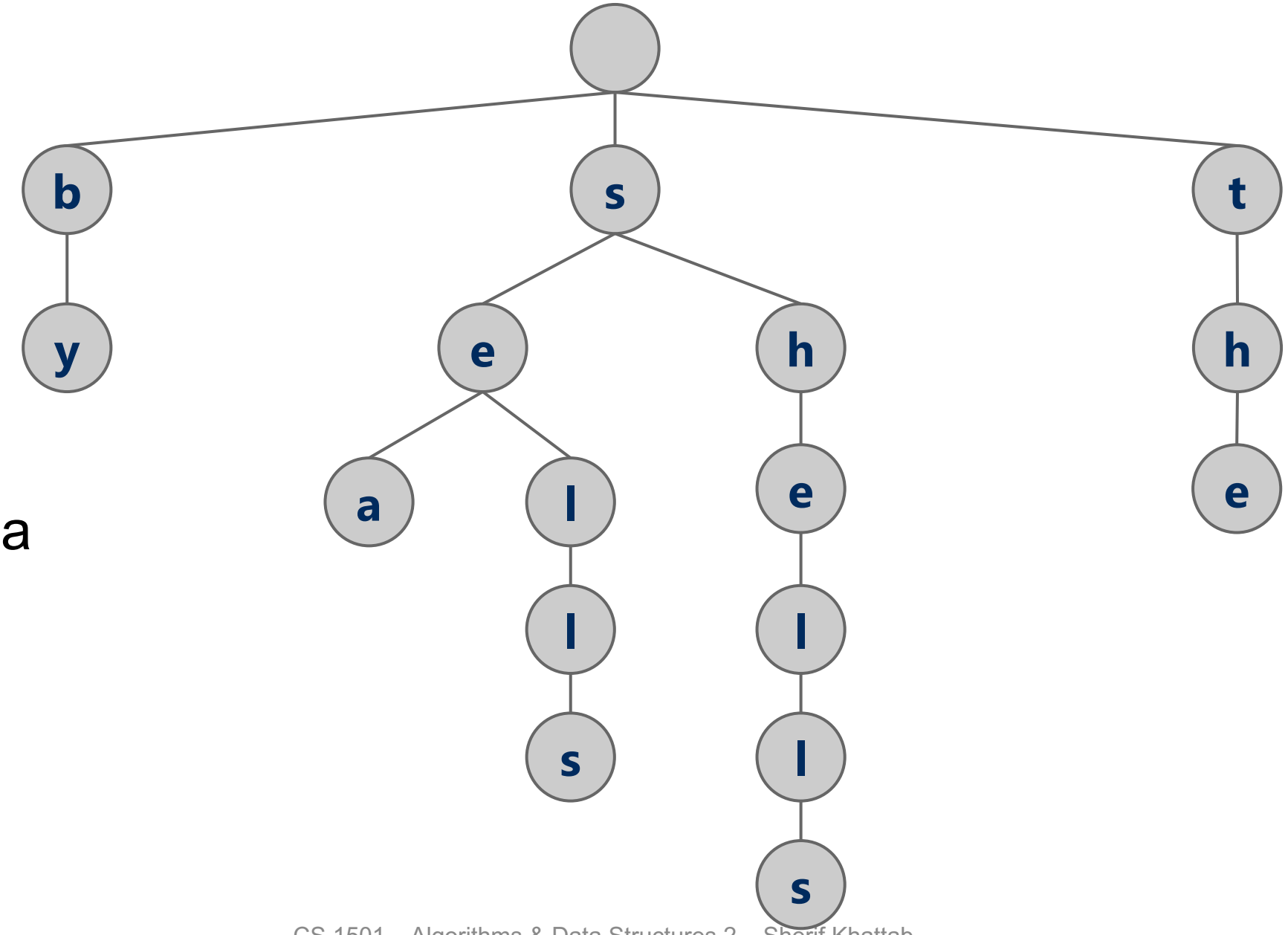
by

Another trie example



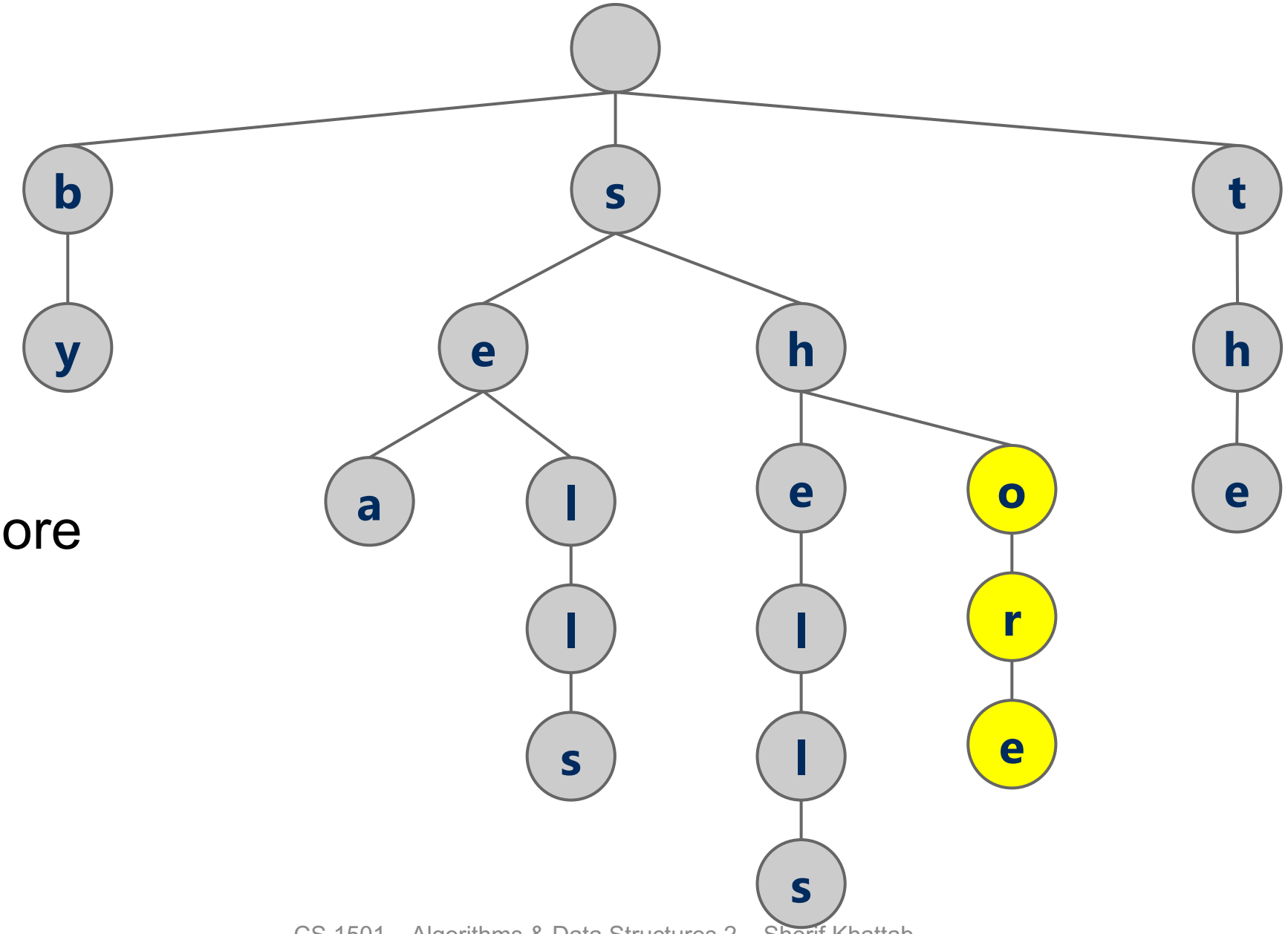
the

Another trie example



sea

Another trie example



shore

Analysis

- Runtime of add and *search hit*?
- O(w) where w is the character length of the string
  - So, what do we gain over RSTs?
    - $w < b$
    - e.g., assuming fixed-size encoding $w = \frac{b}{\lceil \log R \rceil}$
    - tree height is reduced

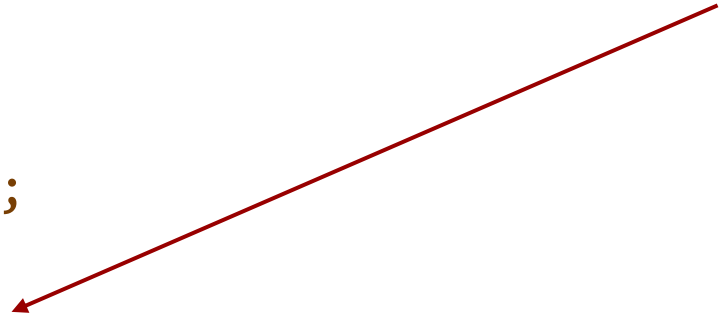
Search Miss

- Search Miss time for R-way RST
  - Require an average of $log_R(n)$ nodes to be examined
    - Proof in Proposition H of Section 5.2 of the text
- Average tree height with 2^{20} keys in an RST?
  - $\log_2 n = \log_2 2^{20} = 20$
- With 2^{20} keys in a large branching factor trie, assuming 8-bits at a time?
  - $\log_R n = \log_{256} 2^{20} = \log_{256} (2^8)^{2.5} = = \log_{256} 256^{2.5} = 2.5$

Implementation Concerns

- See TrieSt.java
  - Implements an R-way trie
- Basic node object:

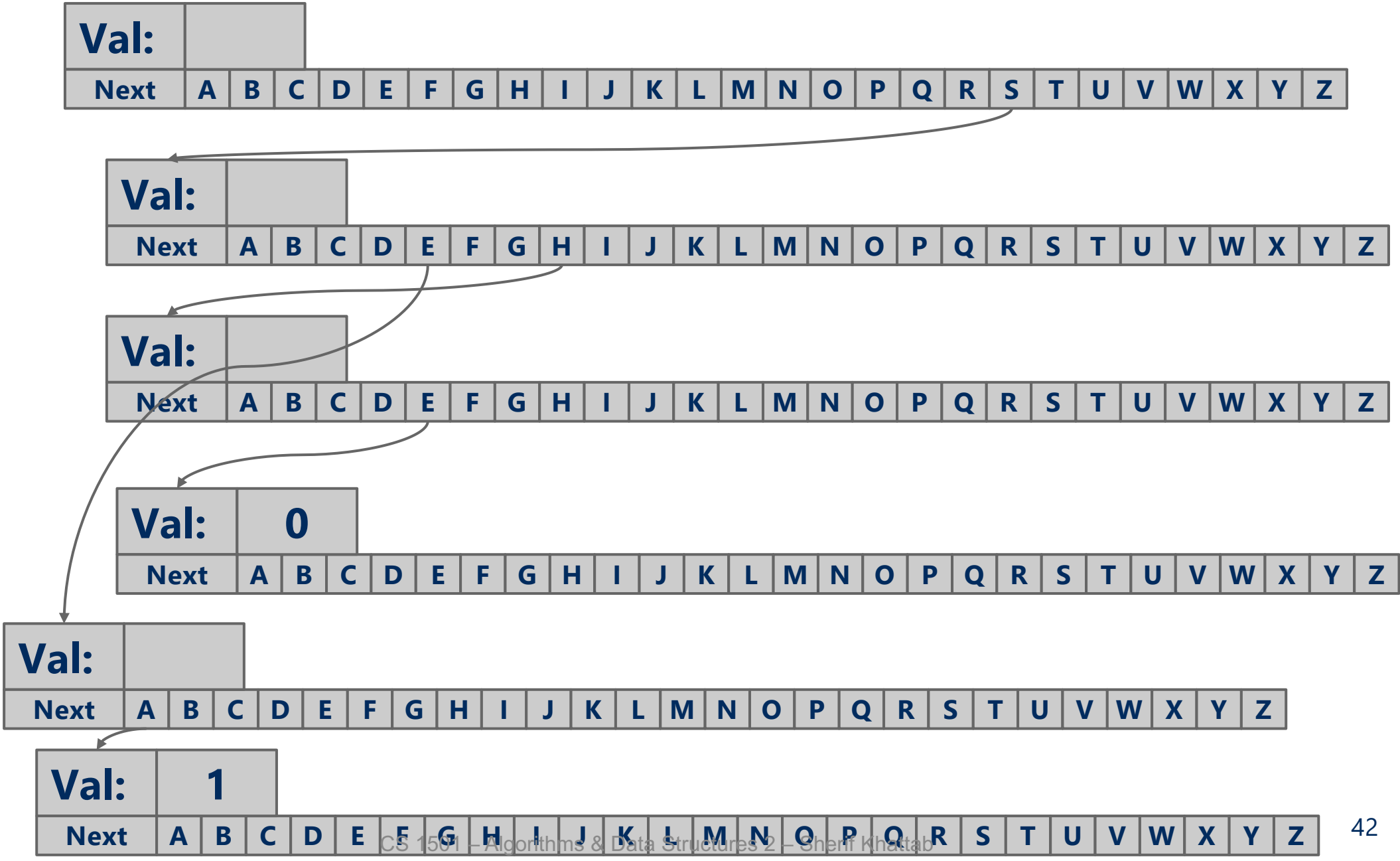
Where R is the branching factor



```
private class Node {
    private Object val;
    private Node[] next;
    private Node(){
      next = new Node[R];
    }
}
```

- Non-null **val** means we have traversed to a valid key
- Again, note that keys are not directly stored in the trie at all

R-way trie example

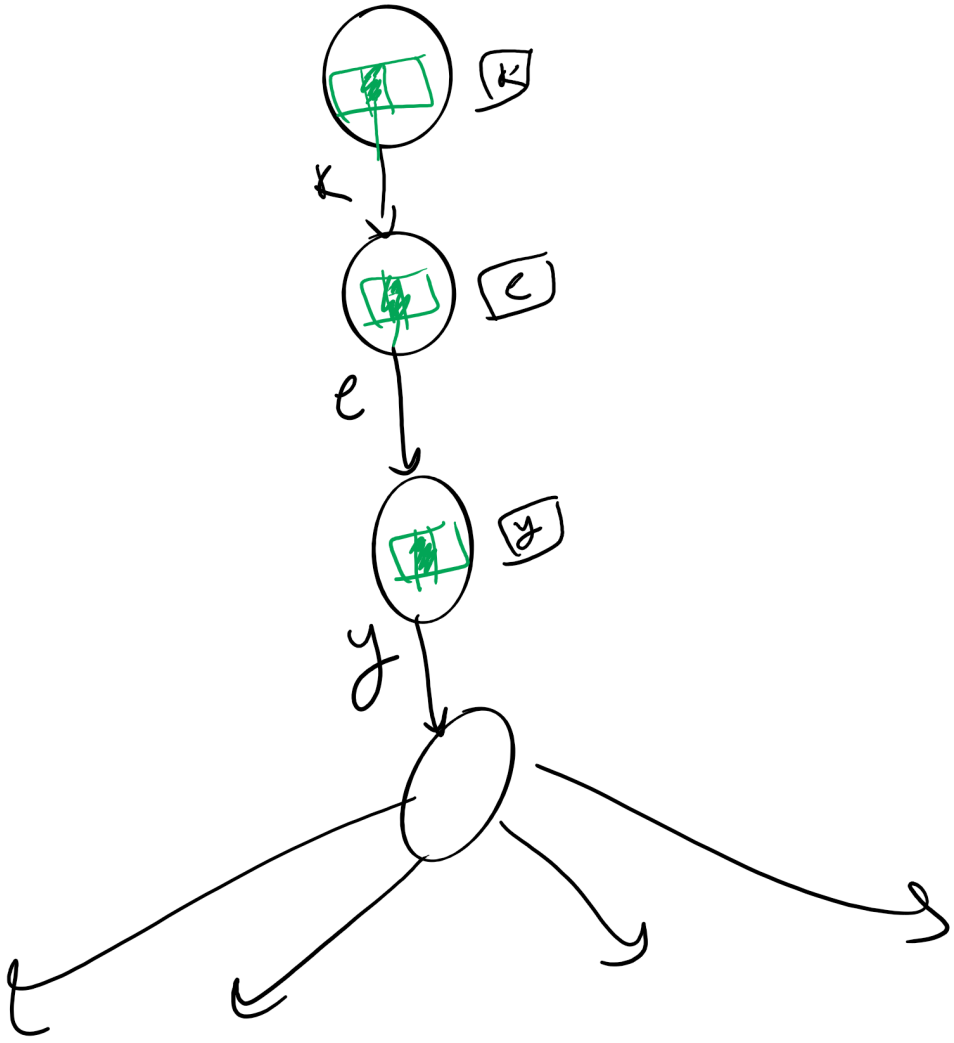


Summary of running time

	insert	search hit	search miss
binary RST	$\Theta(h)$	$\Theta(h)$	$\Theta(\log_2 n)$ on average
multi-way RST	$\Theta(w)$	$\Theta(w)$	$\Theta(\log_R n)$

R-way RST's nodes may waste space!

- Considering 8-bit ASCII, each node contains 2^8 references!
- This is especially problematic as in many cases, a lot of this space is wasted
  - Common paths or prefixes for example, e.g., if all keys begin with "key", thats 255*3 wasted references!
  - At the lower levels of the trie, most keys have probably been separated out and reference lists will be sparse



Solution: De La Briandais tries (DLBs)

Main idea: replace the array inside the node of the R-way trie with a linked-list

De La Briandais (DLB) Trie

- tree-like structure used for searching when keys are sequences of characters
- each nodelet
  - stores one character,
  - points to a sibling (linked list of siblings), and
  - points to a child

Adding to DLB Trie

- if root is null, set root ← new node
- current node ← root
- for each *character c* in the key
  - Search for *c* in the linked list headed at current using sibling links
    - if not found, create a new node and attach as a sibling to the linked list
  - move to child of the found node
    - either recursively or by current ← child
- if at last character of key, insert value into current node and return

DLB Example